Programmation Orientée Objet – Java

Cours 2 : Premiers objets, API Java et Collections

Viviane Pons

Master BIBS Université Paris-Saclay

Premiers Objets

Question

Avez-vous déjà vu un objet ?

Premiers Objets

Question

Avez-vous déjà vu un objet ?

Réponse : OUI, quand vous utilisez un `Scanner`, une chaîne de caractère `String` ou encore `System.out` , vous utilisez des objets !
Mais un objet, c'est quoi ?

Classes et Objets

La classe

```
public class Rational {
    public final int n;
    public final int d;

    Rational(int n, int d) {
        if(d == 0) {
            throw new IllegalArgumentException();
        }
        int div = gcd(n, d);
        n = n / div;
        d = d / div;
        this.n = n;
        this.d = d;
    }


    public Rational add(Rational r2) {
```

Les Objets

```
Rational r1 = new Rational(1,2);
Rational r2 = new Rational(1,3);

Rational r3 = r1.add(r2);
```

Un objet est une **instance** de la classe. Ici, il y a une seule classe `Rational` et 3 instances `r1`, `r2` et `r3`.
Chaque objet a accès à **ses propres champs** (ici `n` et `d`) et à **ses propres méthodes** (ici la méthode `add`).

Méthodes et variables de classe

Déclaration avec `static`

```
public class ExempleStatic {

    public static double temperature;

    public static final double kelvin = 273.15;

    public static double temperatureC() {
        return temperature - kelvin;
    }

    public static double temperatureF() {
        return temperature * 1.8 - 459.67;
    }
}
```

On l'utilise à partir de la classe

```
ExempleStatic.temperature = 300;
System.out.print("Constante Kelvin : ");
System.out.println(ExempleStatic.kelvin);
System.out.print("Température en deg C : ");
System.out.println(ExempleStatic.temperatureC());
System.out.print("Température en deg F : ");
System.out.println(ExempleStatic.temperatureF());
```

Le mot clé `static` pour les champs / méthodes signale les paramètres associés **à la classe** et non **pas aux objets (ou instances)**.

Dans l'exemple, vous voyez qu'on **ne crée pas d'objets**. Les variables et méthodes statiques peuvent être considérées comme des **variables globales** qui sont "rangées" à l'intérieur d'une classe.

Exemples :

- la méthode `main` est toujours `static`.
- la classe `System` ne peut pas être instanciée mais on utilise la variable statique `System.out`

Les objets que vous avez utilisés

Pas des objets :

```
int a = 3;
double x = 1.5;
```

(ce sont des types simples)

Des objets :

```
String s = "bonjour";
Scanner scan = new Scanner(System.in);
```

Mais où sont les classes `String` et `Scanner` ? dans l'API Java

L'API Java

Ce sont l'ensemble des classes qui sont fournies avec Java (plus précisément, avec JRE). Environ 4400 classes organisées en 220 "packages" dans 21 "modules".

Par exemple :

- les classes `String` et `System` se trouvent dans le package `java.lang`
- la classe `Scanner` se trouve dans le package `java.util`

Tous les éléments de `java.lang` sont importés par défaut. Pas ceux de `java.util` ! C'est pour ça qu'on écrit :

```
import java.util.Scanner;
```

Comment connaître ces classes et savoir comment les utiliser ? Réponse : la Javadoc !

La Javadoc

Avez vous remarqué ces petits blocs de commentaire ?

```
/**
 * Teste si une personne est majeure
 * @param age l'age de la personne
 * @return true si la personne est majeure, false sinon
 */
```

Ce sont des blocs de documentation. Ils suivent un formalisme particulier (par exemple `@param` `@return`) et peuvent être utilisés pour **générer des pages automatique de documentation**, c'est le format **Javadoc**.

La documentation **Javadoc** de l'API Java est disponible sur le site d'Oracle

On peut aussi y accéder **directement depuis l'IDE**. Tout comme on peut accéder à **sa propre javadoc**.

Pour la suite

- On utilisera la Javadoc de l'API Java pour trouver les méthodes / classes dont nous avons besoin
- On documentera **nos** classes et **nos** méthodes avec le format Javadoc pour créer une documentation technique automatique.

Application : Tableaux et Collections

Problème courant : gérer un “grand nombre” d’objets

Première solution : les tableaux à taille fixe

Tableaux à taille fixe

Déclaration et création

```
int[] monTableau = new int[10];
int[] monTableau2 = {12,34,25,1};
```

La taille est **fixée** et ne peut plus être modifiée après la création.

Utilisation

```
System.out.println(monTableau.length);
System.out.println(monTableau2.length);

for(int i = 0; i < monTableau2.length; i++) {
    System.out.println(monTableau2[i]);
}

for(int val : monTableau2) {
    System.out.println(val);
}

System.out.println(Arrays.toString(monTableau2));
```

La classe **Arrays** : contient plein de méthodes statiques pour utiliser les tableaux

Exemple deux dimensions (ou plus !)

```
int[][] maMatrice = new int[20][20];
int[][] maMatrice2 = {{1,2,3},{4,5,6}};

for(int i = 0; i < maMatrice2.length; i++) {
    for(int j = 0; j < maMatrice2[i].length; j++ ) {
        System.out.print(maMatrice2[i][j] +  " ");
    }
    System.out.println();
}
```

Mais comment faire quand on veut pouvoir ajouter / supprimer des éléments ?

L'interface Collection et ses classes

Une Interface : qu'est-ce que c'est ?

C'est un **contrat** que doit remplir une classe : une liste de méthode à implanter. On dit qu'une classe **implémente une interface**.

L'interface Collection

Voir la Documentation JavaDoc

Quelques méthodes :

- `add` : pour ajouter un élément
- `addAll` : pour ajouter une collection d'éléments
- `remove` : pour supprimer un élément
- `contains` : est-ce que ma collection contient cet élément ?
- `size` : le nombre d'éléments

L'interface List : une sous-interface de Collection

L'interface `List` contient les méthodes de Collection + d'autres.
Utilisée pour réprésenter une **liste ordonnée d'objets identifiés par un indice** comme pour les tableaux.
Voir la documentation Javadoc
Quelques méthodes :

- `get` : récupérer un objet à un indice donné
- `set` : mettre un objet à un indice donné
- `indexOf` : renvoie l'indice d'un objet donné
- `remove(indice)` : pour supprimer un élément à un indice donné

Comment on crée une liste ?

`Collection` et `List` sont des interfaces, La Javadoc nous indique les classes qui les implémentent. Par exemple `ArrayList`.

Création d'une liste avec `ArrayList` :

```
List<Integer> maListe = new ArrayList<Integer>();
```

Que remarquez vous ?

Comment on crée une liste ?

`Collection` et `List` sont des interfaces, La Javadoc nous indique les classes qui les implémentent. Par exemple `ArrayList`.

Création d'une liste avec `ArrayList` :

```
List<Integer> maListe = new ArrayList<Integer>();
```

Que remarquez vous ?

- Les collections sont écrites de façon **générique**, on doit spécifier le type d'objet entre `<...>`
- Le type **déclaré** (à gauche) n'est pas le type **instancié** (à droite) : c'est le **polymorphisme**
- Qu'est-ce que `Integer` ? Pourquoi pas `int` ? Parce que `int` n'est pas un objet, on utilise un type **wrapper**

Utilisation

Ajouts d'éléments :

```
maListe.add(1);
maListe.add(2);
maListe.add(3);
```

Affichage :

```
System.out.println(maListe);
for(int i = 0; i < maListe.size(); i++) {
    System.out.println(maListe.get(i));
}
for(int v : maListe) {
    System.out.println(v);
}
```

On privilégiera la deuxième boucle (**foreach**) qui fonctionne sur les objets de type `Collection` et tableaux : en effet, les deux implémentent l'interface `Iterable`.

Conversion tableau / list

```
Integer[] monTableau = {1, 2, 3};
List<Integer> maListe2 = Arrays.asList(monTableau);
List<Integer> maListe3 = Arrays.asList(1, 2, 3);
```

Attention, le code suivant ne marche pas !

```
int[] monTableau = {1, 2, 3};
List<Integer> maListe2 = Arrays.asList(monTableau);
```

Mais on peut faire :

```
List<Integer> maListe3 = Arrays.asList(1, 2, 3);
```

Un “détail” très important

En Java, les variables stockent non pas les objets mais **la référence à l’objet** – i.e. l’adresse mémoire.

Qu’affiche ce code ?

```
List<Integer> maListe = Arrays.asList(1, 2, 3);
List<Integer> maListe2 = maListe;
maListe.set(0, 4);
System.out.println(maListe2);
```

Un “détail” très important

En Java, les variables stockent non pas les objets mais **la référence à l'objet** – i.e. l'adresse mémoire.

Qu'affiche ce code ?

```
List<Integer> maListe = Arrays.asList(1, 2, 3);
List<Integer> maListe2 = maListe;
maListe.set(0, 4);
System.out.println(maListe2);
```

```
[4, 2, 3]
```

Pourquoi ? `maListe` et `maListe2` sont des **références** au même objet en mémoire.

Solution :

```
maListe = Arrays.asList(1, 2, 3);
maListe2 = List.copyOf(maListe);
maListe.set(0, 4);
System.out.println(maListe2);
```

Qu'affiche ce code ?

```
List<Integer> maListe = Arrays.asList(1, 2, 3);
List<Integer> maListe2 = Arrays.asList(1, 2, 3);
System.out.println(maListe == maListe2);
```

Qu'affiche ce code ?

```
List<Integer> maListe = Arrays.asList(1, 2, 3);
List<Integer> maListe2 = Arrays.asList(1, 2, 3);
System.out.println(maListe == maListe2);
```

```
false
```

Pourquoi ? Le == teste l'égalité des références mémoires ! Ici, on a deux instances différentes.

Solution :

```
System.out.println(maListe.equals(maListe2));
```

Que se passe-t-il quand on ne crée pas l'objet ?

```
List<Integer> maListe;
```

Par défaut, `maListe` est une "référence vide", c'est un pointeur vers rien du tout qui en Java prend la valeur `null`. Si vous essayez de l'utiliser, une erreur `NullPointerException` apparaîtra.

D'autres Interfaces utiles :

- Les interfaces `Set` et `SortedSet` (sous-interfaces de `Collection`) implantées par exemple avec `HashSet` et `TreeSet` : les ensembles contenants les objets une unique fois.
- L'interface `Map<K,V>` souvent implémentée par `HashMap` : pour stocker des "dictionnaires" clés – valeurs.

Assez discuter, au travail !

Le TP